OS-Level AI-Orchestrated Post-Quantum Key Fabric With OTP-Like Session Properties

Distributed ML-KEM (CRYSTALS-Kyber) bootstrapping, multi-vantage key agreement, and high-rate ratcheting at kernel boundaries

Author: Paul George Savluc
Date: January 02, 2026
Keywords: post-quantum cryptography, CRYSTALS-Kyber, ML-KEM, operating system encryption, horizontal scaling, distributed key management, key transparency, zero trust, OTP-like keystream, forward secrecy, device mesh security, kernel networking security, AI security orchestration, entropy injection, sensor-derived rekeying

Abstract. This document specifies an operating system level encryption architecture that scales horizontally across large fleets of connected devices while resisting man in the middle (MITM) substitution, passive interception, and single backend key exposure. The design uses ML-KEM (standardized from CRYSTALS-Kyber) to bootstrap post-quantum secrecy, authenticates handshake transcripts to prevent active substitution, and derives traffic keys from a multi-vantage key agreement where multiple geographically and administratively separated nodes contribute independent secrets. Above the bootstrapping layer, an OS resident key fabric performs high rate symmetric ratcheting to produce OTP-like session properties: each message uses a unique, nonrepeating keystream segment derived from continuously evolving state, with strict nonce discipline and replay resistant transcript binding. An AI control plane administers policy and enforcement at the system boundary without ever gaining access to exportable private keys or raw session material. Optional entropy injection can incorporate device-local sensor noise and timing jitter, including a formalized observation surface model that hashes spatiotemporal measurements without relying on semantic scene labels. A Fibonacci timed rotation policy and a golden ratio torus scheduler are introduced as deterministic, low overhead mechanisms for unpredictable node quorum selection and rekey cadence, improving interception resistance without replacing cryptographic authentication.

Key contributions.

- OS level enforcement below applications for traffic, storage, and device to device links.
- Post-quantum bootstrapping with ML-KEM plus authenticated transcripts and key confirmation.
- Multi-vantage worldwide secret contribution and threshold governed notary mesh to avoid single backend exposure.
- OTP-like keystream segments via strict nonce discipline and symmetric ratcheting at kernel boundaries.
- AI administered policy and telemetry with nonexportable keys and capability tokens, keeping AI out of the key custody path.
- Optional observation surface entropy injection that is independent of semantic computer vision.

Contents

1. Scope and non goals
2. Threat model and security properties
3. True OTP versus OTP-like in deployed systems
4. System architecture: OS Key Fabric (OKF)
5. Identity, enrollment, and attestation
6. AI administration model and authority boundaries
7. Post-quantum bootstrapping: ML-KEM plus authenticated transcripts
8. Multi-vantage worldwide key agreement and horizontal scaling
9. Symmetric ratcheting, keystream segments, and OS interfaces
10. Observation surface entropy injection (spacetime model)
11. Fibonacci rotation and phi torus scheduling (dimensional encapsulation)
12. OS integration: Linux, Windows, embedded
13. Security analysis and failure modes
14. Performance, operations, and rollout plan
15. Reference pseudocode and implementation notes
References

1. Scope and non goals

This document targets system level encryption that can be enforced by the operating system across a fleet of connected devices. The goal is to protect data in motion and, optionally, data at rest using a shared key fabric that is administered centrally as policy but executed locally with nonexportable keys. The architecture assumes heterogeneous devices: servers, desktops, mobile, and embedded controllers.

Non goals. No cryptographic design can guarantee confidentiality if endpoints are fully compromised. The design does not claim to prevent malware on a device from reading plaintext after decryption. It also does not claim unconditional one time pad security without explicit pad distribution. Instead, it targets practical OTP-like session properties under realistic fleet threat models.

2. Threat model and security properties

Adversaries. The system is designed to withstand passive eavesdroppers who record traffic, active MITM adversaries who modify traffic, and partial infrastructure compromise, including compromise of one region, one notary node, one provider account, or one device. The design also considers route manipulation and selective packet injection.
Required properties.

- Authenticated key establishment: prevent silent key substitution and enforce key confirmation.
- Post-quantum confidentiality: resist harvest now, decrypt later collection using ML-KEM bootstrapping.
- Forward secrecy: compromise of long term keys does not reveal past traffic keys.
- Compartmentalization: no single backend or region holds enough to decrypt fleet traffic.
- Horizontal scaling: adding devices increases capacity without introducing a central decryption bottleneck.
- Crypto agility: rotate algorithms and keys rapidly under policy.
- Replay resistance: time and counter binding at OS boundary; reject replays and out of window packets.

3. True OTP versus OTP-like in deployed systems

True OTP is information theoretic only when the pad is truly random, at least as long as the message, never reused, and shared only by the legitimate parties. Fleet scale OTP fails on distribution: distributing pads at scale without introducing a central pad holder is operationally difficult.

Practical target. OKF aims for OTP-like behavior by guaranteeing nonrepetition of keystream segments (unique key and nonce for each message) and by evolving secret state so compromise does not retroactively reveal prior keystream. This matches the security posture of modern ratcheting protocols, but implemented at the OS boundary.

When true OTP is possible. If a fleet can physically preload pads or use a secure

manufacturing pipeline to inject pads into devices, OKF can consume those pads as entropy to approach true OTP properties. This document includes a pad consumption interface, but does not assume pad distribution is always available.

4. System architecture: OS Key Fabric (OKF)

OKF is a layered system spanning kernel, user space, and fleet governance. It is composed of five planes:

(A) Kernel crypto boundary. Attaches to network and storage paths and enforces encryption.

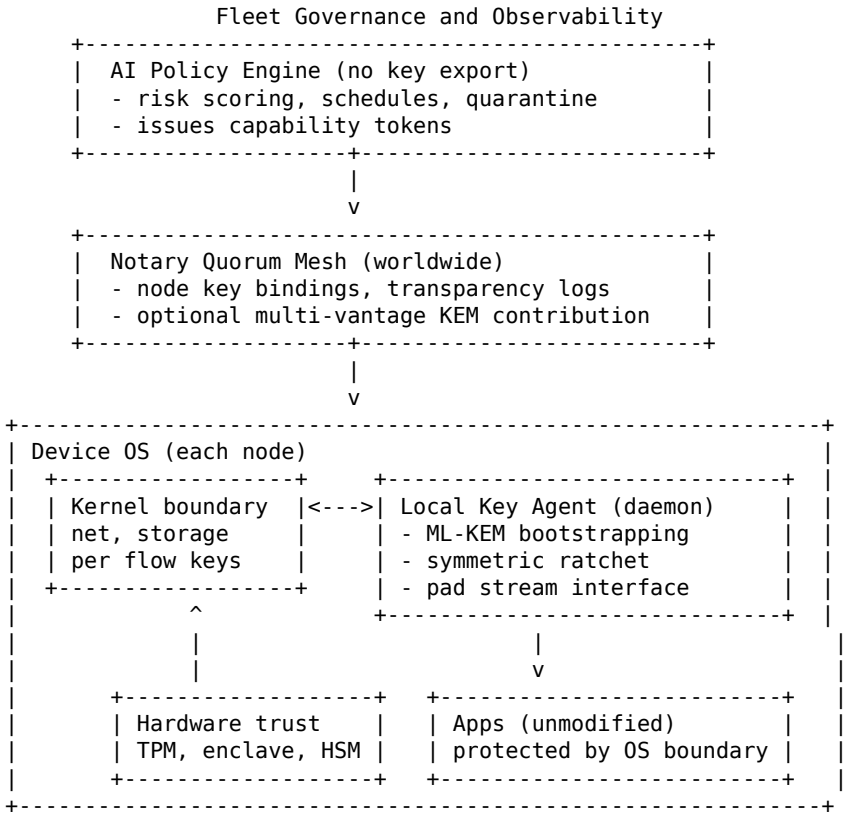
(B) Local key agent. Privileged daemon performing handshakes, ratcheting, key derivation, and key installation.

(C) Notary quorum mesh. Globally distributed nodes providing authenticated key bindings, optional secret contribution, and transparency logging.

(D) Hardware root of trust. TPM, enclave, HSM, or secure element storing long lived private keys as nonexportable.

(E) AI policy engine. Monitoring and orchestration that schedules rotation and issues signed capability tokens, without key custody.

Figure 1. OS Key Fabric high level component map.



5. Identity, enrollment, and attestation

Every device is a cryptographic principal. Each device has a hardware protected identity key and a measured software baseline.

Enrollment. A device generates a key pair inside TPM or enclave and registers the public key with governance. Governance issues a device certificate or a signed device record and writes it to a transparency log.

Attestation gating. Admission to the fabric is conditioned on attestation evidence. Policies can require measured boot, agent version, kernel module hash, and security posture signals.

Key nonexportability. Long lived private keys are marked nonexportable. The local key agent requests signing or decapsulation operations from hardware, but cannot read private key bytes.

6. AI administration model and authority boundaries

The AI subsystem is treated as a policy engine that issues decisions and schedules, not as a cryptographic principal. If an AI can export keys, it becomes the single backend. OKF explicitly prevents this by construction.

Capability tokens. The AI policy engine issues short lived signed tokens authorizing specific actions. Tokens include device identity, scope, expiration, and a nonce. The local agent validates tokens and executes allowed operations.

Least authority. The AI can: approve rotation, adjust quorum size, quarantine devices, and require additional attestation. The AI cannot: export private keys, export traffic keys, or disable transcript authentication. Hardware and kernel interfaces enforce these constraints.

6.1 Example AI driven events

- Route anomaly detected: accelerate Fibonacci epoch reset and force ML-KEM refresh.
- Transparency log inconsistency: remove affected notary nodes from quorum and quarantine related sessions.
- Device attestation failure: revoke capability tokens and drop kernel key installs for that device.
- Key reuse suspicion: rotate flow keys and lock counters; trigger incident workflow.

7. Post-quantum bootstrapping: ML-KEM plus authenticated transcripts

ML-KEM provides post-quantum secrecy against passive recording. Authentication closes active MITM. The handshake transcript is hashed and signed, and then confirmed by MAC.

Transcript fields. identities, certificates, capability token hashes, algorithm suite, nonces, counters, time window, selected notary set, attestation summary hashes, and ciphertext hashes.

Key confirmation. Both sides exchange a confirmation MAC derived from the boot secret and computed over the transcript hash. Any tampering causes failure.

8. Multi-vantage worldwide key agreement and horizontal scaling

Instead of one key exchange path, OKF optionally uses k independent ML-KEM contributions from globally distributed nodes. The final boot secret is derived by combining all component secrets through HKDF.

Worldwide hardness. A passive tap at one location only sees one path. Even multiple taps must capture every component contribution. A compromise of one notary node leaks at most one component secret and does not yield traffic keys.

Threshold governance. Notary nodes are governed by threshold rules: adding or removing nodes, or changing quorum sizes, requires signatures from multiple administrators or multiple independent control domains. This reduces insider and single account risks.

9. Symmetric ratcheting, keystream segments, and OS interfaces

After bootstrapping, the OS should remain in the symmetric fast path. The local agent maintains ratchet state per peer and per lane. The kernel boundary requests traffic keys for a flow tuple and epoch.

Kernel interface contract. The kernel never receives long lived state. It receives only short lived traffic keys, nonce derivation rules, and replay window parameters. Counters are maintained in kernel to guarantee uniqueness.

Virtual pad interface. For systems that want pad semantics, OKF can provide a character device that streams keystream bytes. Each read consumes bytes from a unique counter indexed segment. This is pad-like behavior backed by a ratchet. If a true pre-shared pad exists, it can be mixed as additional entropy or used directly as keystream.

9.1 Nonce discipline checklist

- Nonce must be unique per key. Enforce in kernel, not in applications.
- Use monotonic per flow counters and bind direction bit to nonce construction.
- Persist counters across reboot or re-establish new keys on reboot.
- Reject replays by maintaining a sliding window for inbound sequence numbers.

10. Observation surface entropy injection (spacetime model)

This section formalizes the idea that a device can incorporate a spatiotemporal observation surface into key evolution without relying on semantic scene understanding. The objective is not to treat vision as the sole secret, but to add device-local unpredictability that does not traverse the network.

Observation surface. At time t, the device captures a measurement vector X(t) consisting of raw sensor bytes and time tags. Example components: camera sensor noise, IMU readings, microphone noise floor summaries, clock jitter, and a monotonic counter. The system computes e(t) = H(domain || X(t)).

Ratcheting with injection. state_{i+1} = HKDF(state_i || e(t_i) || i, salt = transcript_hash, info = 'ratchet-step').

Anti spoofing notes. Semantics such as tree or television brand are discarded. The system favors raw noise components that are difficult to predict remotely. Rendered or virtual scenes can be included, but should be treated as adversary influenceable. Mixing hardware noise mitigates this risk.

10.1 Dimensions and measurement packing

- Time dimension: coarse wall clock plus monotonic counter and jitter measurements.
- Spatial proxies: IMU orientation deltas and camera readout timing skew.
- Observation bytes: raw frame sensor residuals after subtracting a low-pass estimate, not semantic features.
- Environmental noise: audio band energy summaries and thermal sensor readings.
- Device identity: bind to device id hash so measurements cannot be replayed across devices.

11. Fibonacci rotation and phi torus scheduling (dimensional encapsulation)

The phrase encapsulate keys within a phi torus is implemented here as a deterministic scheduler over a 2-torus that drives quorum selection and lane selection. It is not a new cryptographic primitive. It is an operational mixing function that changes which components contribute to keys over time.

Fibonacci timing. Rekey intervals follow interval(n) = clamp(F(n) * T0, min, max). The AI can reset n based on risk signals, compressing time to force rapid churn.

Phi torus embedding. Let phi be the golden ratio. For step index i and denominator D, define u = frac(i / D) and v = frac(phi * i / D). Map to angles theta = 2*pi*u and psi = 2*pi*v. The pair (theta, psi) indexes a quasi periodic point on a torus. Hashing node identifiers with (theta, psi) yields a deterministic but difficult to predict selection across time.

Multi dimensional extension. Extend to higher dimensions by using additional irrational multipliers such as sqrt(2) and sqrt(3) to generate more phases. Use these phases only

for scheduling and selection, not as a replacement for cryptographic randomness.

12. OS integration: Linux, Windows, embedded

OKF is designed to be OS native. Applications should not be modified. The OS boundary enforces encryption for traffic and storage, and the local agent maintains state.

12.1 Linux integration

- Network: kernel TLS (kTLS) or IPsec (XFRM). Keys injected by local agent.
- Storage: dm-crypt or fscrypt using keys derived by OKF.
- Key API: netlink or character device. Kernel stores counters and replay windows.
- Hardening: seccomp, SELinux or AppArmor confinement for the agent.

12.2 Windows integration

- Network: IPsec plus WFP callouts or kernel drivers enforcing encryption per flow.
- Crypto: CNG with TPM protected keys and restricted service identities.
- Agent kernel channel: IOCTL interface with strict ACLs and audit logging.
- Attestation gating: device health controls access to capability tokens.

12.3 Embedded and IoT integration

- Use a small agent with ML-KEM bootstrapping and symmetric ratchet.
- Secure element storage for identity keys when TPM is unavailable.
- Batch KEM work and rely on symmetric rekey to minimize overhead.
- AEAD hardware acceleration where available.

13. Security analysis and failure modes

MITM. Prevented by transcript authentication and key confirmation. Scheduling complexity does not replace authentication.

Passive taps. Post-quantum bootstrapping protects against harvest now, decrypt later. Multi-vantage contributions raise the bar for interception by requiring multiple paths.

Partial compromise. Compromise of a notary node leaks at most one component. Compromise of one device leaks only that device's plaintext and local secrets.

Operational failures. The most dangerous failure mode is nonce reuse. OKF therefore pushes counter ownership into kernel and forces key refresh on reboot or counter uncertainty.

AI failure. AI errors cannot directly leak keys due to nonexportability. AI can still cause outages if it quarantines incorrectly, so governance should include human override and audit trails.

14. Performance, operations, and rollout plan

Most cycles belong in symmetric crypto. ML-KEM is used for bootstrapping and infrequent refresh.

Rollout phases. (1) Start with IPsec or kTLS enforcement and a local agent performing symmetric key rotation only. (2) Add ML-KEM bootstrapping and transcript authentication. (3) Add multi-vantage contributions and transparency logging. (4) Add optional observation surface entropy injection.

Observability. Export only metadata: rotation epochs, handshake success rates, transparency consistency, and counter health. Never export raw keys or raw sensor bytes.

15. Reference pseudocode and implementation notes

The following pseudocode sketches a minimal OKF. It is intentionally implementation oriented, describing the interfaces between AI policy, local agent, and kernel boundary.

Listing 1. Minimal OKF pseudocode.

```
# Pseudocode: OS Key Fabric per peer state

on_enroll(device):
    generate identity key inside TPM or enclave (nonexportable)
    register public key and attestation policy with governance
    write device record into transparency log

on_open_flow(peer_id, flow_tuple):
    if no active session with peer_id:
        run_pq_bootstrap(peer_id)
    flow_key = KDF(session_state, flow_tuple, direction, epoch)
    install flow_key into kernel boundary
    kernel initializes per flow counter and replay window
    return

run_pq_bootstrap(peer_id):
    token = request_capability_token(peer_id)
    nodes = select_quorum_nodes(peer_id, time, fib_epoch, phi_torus)

    for node in nodes:
        pk_node, binding = fetch_node_key(node)
        verify binding via policy and transparency
        ct_node, ss_node = ML-KEM-Encaps(pk_node)
        send ct_node to peer and record ciphertext hash

    thash = H(identities || token_hash || nonces || counters || algs || nodes || bindings || cip
hertext hashes)
    ss_boot = HKDF(ss_1 || ... || ss_k, salt=thash, info="kem-combine")
    state_0 = HKDF(ss_boot, salt=thash, info="ratchet-init")

    confirm = HMAC(KDF(state_0, "confirm"), thash)
    exchange confirm and verify, else abort

ratchet_step(peer_id):
    entropy = optional_observation_surface_hash()
    state = HKDF(state || entropy || counter, salt=thash, info="ratchet-step")
    return state

encrypt_packet(peer_id, plaintext, seq):
    msg_key = HKDF(state || seq, salt=thash, info="msg-key")
    nonce = kernel_derived_unique_nonce(peer_id, seq, direction)
    return AEAD_Encrypt(msg_key, nonce, aad=header, plaintext)
```

15.1 OTP-like checklist for auditors

A system can claim OTP-like session properties only if it can demonstrate nonrepetition of keystream segments. Auditors should verify:

1. Nonce uniqueness is enforced in kernel.
2. Keys are rotated or ratcheted frequently enough to bound exposure.
3. Replay windows and sequence counters are correct and durable.
4. Key separation is explicit via KDF info strings.
5. AI cannot export keys and cannot disable authentication.

6. Multi-vantage selection does not collapse to a single region by accident.

Signed,
Paul George Savluc

References

1. National Institute of Standards and Technology. FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard (ML-KEM). 2024.

2. National Institute of Standards and Technology. FIPS 204: Module-Lattice-Based Digital Signature Standard (ML-DSA). 2024.

3. IETF. RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3. 2018.

4. IETF. RFC 5869: HMAC-based Extract-and-Expand Key Derivation Function (HKDF). 2010.

5. IETF. RFC 8439: ChaCha20 and Poly1305 for IETF Protocols. 2018.

6. IETF. RFC 4301: Security Architecture for the Internet Protocol. 2005.

7. B. Laurie, A. Langley, E. Kasper. Certificate Transparency. 2013.

8. M. Melara et al. CONIKS: Bringing Key Transparency to End Users. 2015.

9. Open Quantum Safe project. liboqs and associated tooling for post-quantum KEM and signature prototyping.